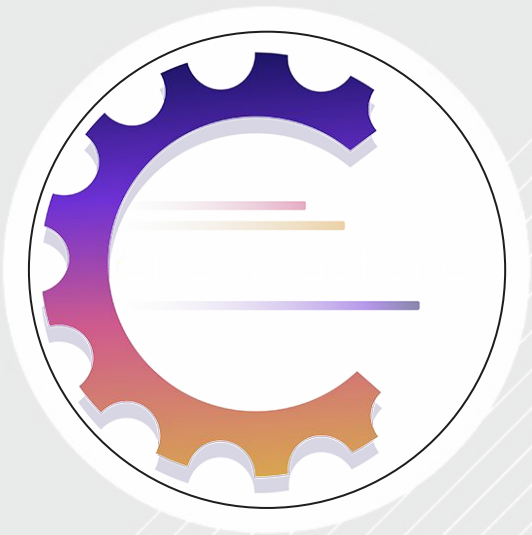
SPYWOLF

Security Audit Report

(TESTNET)



Audit prepared for

Chain Factory

Completed on

April 19, 2024

OVERVIEW

This goal of this report is to review the main aspects of the project to help investors make an informative decision during their research process.

You will find a a summarized review of the following key points:

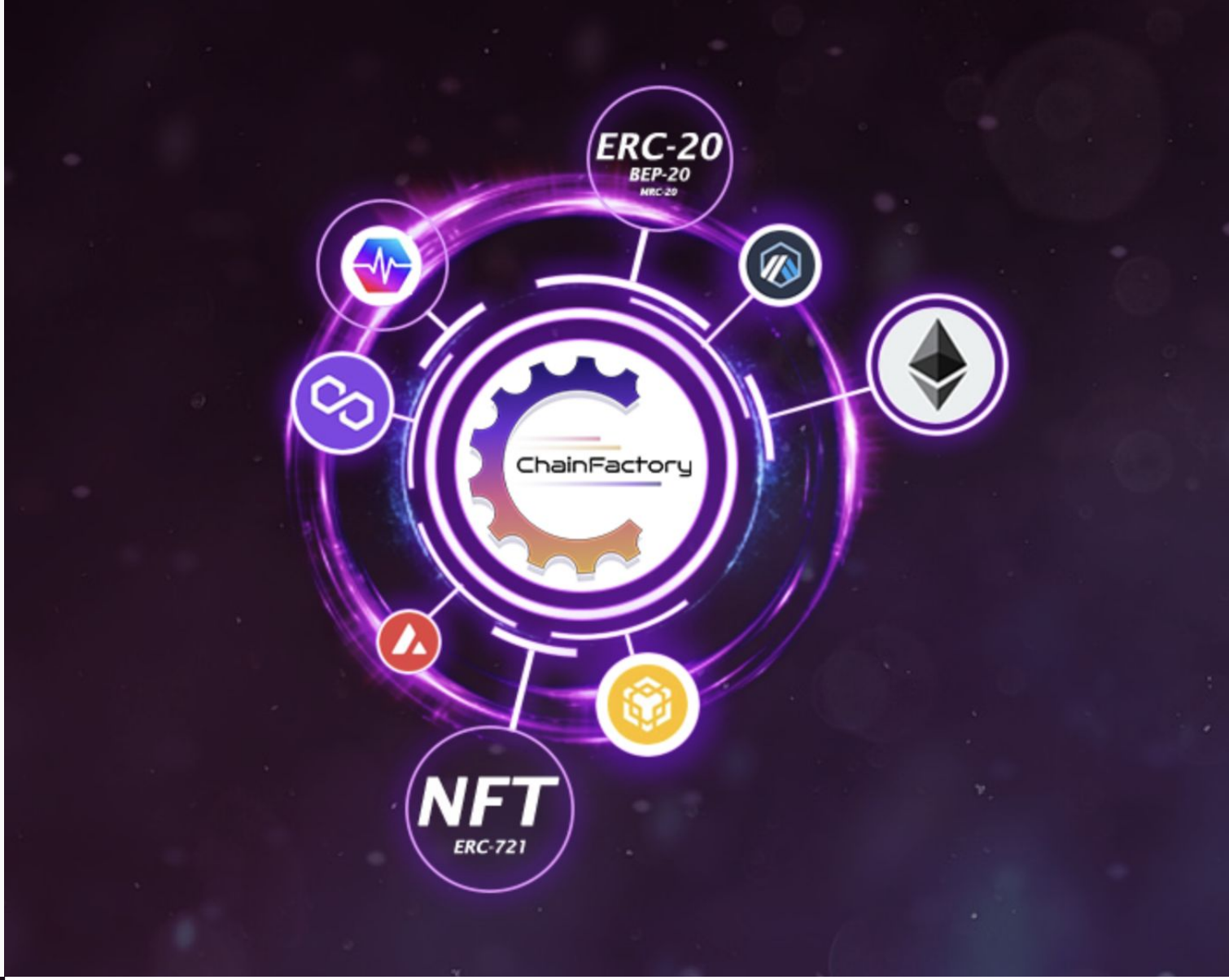
- ✔ Contract's source code
- ✔ Owners' wallets
- ✔ Tokenomics
- ✔ Team transparency and goals
- ✔ Website's age, code, security and UX
- ✔ Whitepaper and roadmap
- ✔ Social media & online presence

The results of this audit are purely based on the team’s evaluation and does not guarantee nor reflect the projects outcome and goal

- SPYWOLF Team -

TABLE OF CONTENTS

CHAIN FACTORY



PROJECT DESCRIPTION

"With ChainFactory, users can choose from a variety of customizable templates and features, making it simple to create contracts tailored to your specific needs. It is designed to be user-friendly and intuitive, guiding users through the entire process step-by-step, providing a centralized platform to create, deploy, and manage your Smart-Contracts with ease."

Release Date: TBD

Category: Ecosystem

CONTRACT INFO

Token Name
N/A

Symbol
N/A

Contract Address
0xBeEC9EB1363d9F60BdE06Ef15472c0D1EDBdCA7a

Network
Ethereum Sepolia TESTNET

Language
Solidity

Deployment Date
Apr 15, 2024

Contract Type
Staking

Total Supply
N/A

Status
Not launched

TAXES

Buy Tax
none

Sell Tax
none

*Taxes can be changed in future

Our Contract Review Process

The contract review process pays special attention to the following:

- Testing the smart contracts against both common and uncommon vulnerabilities
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Blockchain security tools used:

- OpenZeppelin
- Mythril
- Solidity Compiler
- Hardhat

CURRENT STATS

(As of April 17, 2024)

Liquidity

Not added yet

Burn

No burnt tokens

Status:

Not Launched!

MaxTxAmount

N/A

LP Address(es)

Liquidity not added yet

TOKEN TRANSFERS STATS

Transfer Count	N/A
Uniq Senders	N/A
Uniq Receivers	N/A
Total Amount	N/A
Median Transfer Amount	N/A
Average Transfer Amount	N/A
First transfer date	N/A
Last transfer date	N/A
Days token transferred	N/A

SMART CONTRACT STATS

Calls Count	N/A
External calls	N/A
Internal calls	N/A
Transactions count	N/A
Uniq Callers	N/A
Days contract called	N/A
Last transaction time	N/A
Created	N/A
Create TX	N/A
Creator	N/A

VULNERABILITY ANALYSIS

ID	Title	
SWC-100	Function Default Visibility	**Passed**
SWC-101	Integer Overflow and Underflow	**Passed**
SWC-102	Outdated Compiler Version	**Passed**
SWC-103	Floating Pragma	**Passed**
SWC-104	Unchecked Call Return Value	**Passed**
SWC-105	Unprotected Ether Withdrawal	**Passed**
SWC-106	Unprotected SELFDESTRUCT Instruction	**Passed**
SWC-107	Reentrancy	**Passed**
SWC-108	State Variable Default Visibility	**Passed**
SWC-109	Uninitialized Storage Pointer	**Passed**
SWC-110	Assert Violation	**Passed**
SWC-111	Use of Deprecated Solidity Functions	**Passed**
SWC-112	Delegatecall to Untrusted Callee	**Passed**
SWC-113	DoS with Failed Call	**Passed**
SWC-114	Transaction Order Dependence	**Passed**
SWC-115	Authorization through tx.origin	**Passed**
SWC-116	Block values as a proxy for time	**Passed**
SWC-117	Signature Malleability	**Passed**
SWC-118	Incorrect Constructor Name	**Passed**

VULNERABILITY ANALYSIS

ID	Title	
SWC-119	Shadowing State Variables	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed
SWC-121	Missing Protection against Signature Replay Attacks	Passed
SWC-122	Lack of Proper Signature Verification	Passed
SWC-123	Requirement Violation	Passed
SWC-124	Write to Arbitrary Storage Location	Passed
SWC-125	Incorrect Inheritance Order	Passed
SWC-126	Insufficient Gas Griefing	Passed
SWC-127	Arbitrary Jump with Function Type Variable	Passed
SWC-128	DoS With Block Gas Limit	Passed
SWC-129	Typographical Error	Passed
SWC-130	Right-To-Left-Override control character (U+202E)	Passed
SWC-131	Presence of unused variables	Passed
SWC-132	Unexpected Ether balance	Passed
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Passed
SWC-134	Message call with hardcoded gas amount	Passed
SWC-135	Code With No Effects	Passed
SWC-136	Unencrypted Private Data On-Chain	Passed

THREAT LEVELS

When performing smart contract audits, our specialists look for known vulnerabilities as well as logical and access control issues within the code. The exploitation of these issues by malicious actors may cause serious financial damage to projects that failed to get an audit in time. We categorize these vulnerabilities by the following levels:

High Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Medium Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

Low Risk

Issues on this level are minor details and warning that can remain unfixed.

Informational

Information level is to offer suggestions for improvement of efficacy or security for features with a risk free factor.

FOUND THREATS

⚠ High Risk

If **taxDeposit** is higher than the amount deposited by user and contract has enough balances to transfer tax amount to the treasury wallet, overflow will occur in 'amount' variable.
This can cause further math errors in other functions (like calculateRewards()) and artificially inflated staking amount in favour of the user for up to uin256 max value - the reminder of (amount - tax).

```
function deposit(uint256 poolId, uint256 amount) external isPool(poolId) nonReEntrant {
Pool storage pool = _pool[poolId];

unchecked {
    if (pool.taxDeposit > 0) {
    uint256 tax = _percentage(amount, uint256(pool.taxDeposit));

    amount -= tax;
    pool.collectedTaxes += tax;

    if (_treasury != address(0)) {
      bool txFee = IERC20(pool.stakedToken).transfer(_treasury, tax);
      require(txFee, "Transfer error");
    }
}
.....................
    _user[msg.sender].totalDeposited[poolId] += amount;
    _user[msg.sender].totalStaked[poolId] += amount;
    _user[msg.sender].list[poolId].push(stakeId);

    _pool[poolId].totalDeposited += amount;
    _pool[poolId].totalStaked += amount;
.....................
}
}

function calculateRewards(address user, uint256 poolId,
uint256 stakeId) public view returns (uint256 reward) {
    Pool storage pool = _pool[poolId];
    Stake memory stake = _user[user].pool[poolId][stakeId];

    unchecked {
      uint32 timeElapsed = (stake.lockedUntil > 0 ?
      (stake.lockedUntil > _timestamp() ? _timestamp() : stake.lockedUntil) : _timestamp())
      - stake.lastAction;

      reward = (stake.amount * uint256(pool.apy) * uint256(timeElapsed))
      / (uint256(_denominator) * uint256(pool.rewardPeriod));
    }
}
```

- Recommendation:
  - Ensure that the corresponding taxes will never exceed 100% to prevent overflow.

FOUND THREATS

⚠ High Risk

If **taxClaim** is higher than the reward amount and contract have enough balances to transfer tax amount to the treasury wallet, overflow will occur in 'amount' variable, causing claim function to revert and/or drain the entire pool if enough tokens are presented.

```
function claim(uint256 poolId, uint256 stakeId)
external isUser(msg.sender) isPool(poolId) isStake(poolId, stakeId) nonReEntrant {
    _claim(msg.sender, poolId, stakeId);
}

function _claim(address user, uint256 poolId, uint256 stakeId) private {
Pool storage pool = _pool[poolId];
Stake storage stake = _user[user].pool[poolId][stakeId];

require(stake.amount > 0, "Withdrawn");

unchecked {
  require(stake.lastAction + pool.claimCooldown <= _timestamp(), "Claim cooldown");

  uint256 amount = calculateRewards(user, poolId, stakeId);

  if (pool.taxClaim > 0) {
    uint256 tax = _percentage(amount, uint256(pool.taxClaim));

    amount -= tax;
    pool.collectedTaxes += tax;

    if (_treasury != address(0)) {
      bool txFee = IERC20(pool.rewardToken).transfer(_treasury, tax);
      require(txFee, "Transfer error");
    }
  }

  stake.lastAction = _timestamp();

  _user[user].totalRewardsClaimed[poolId] += amount;
  _pool[poolId].totalRewardsClaimed += amount;

  IERC20(pool.rewardToken).transfer(user, amount);

  emit Claim(user, poolId, stakeId, amount);
}
}
```

- Recommendation:
  - Ensure that the corresponding taxes will never exceed 100% to prevent overflow.

FOUND THREATS

⚠ High Risk

If **taxCompound** is higher than the rewards amount and contract have enough balances to transfer tax amount to the treasury wallet, overflow will occur in 'rewards' variable, causing user's stake amount to be artificially inflated.

```
function compound(uint256 poolId, uint256 stakeId) external isUser(msg.sender)
isPool(poolId) isStake(poolId, stakeId) nonReEntrant {
    _compound(msg.sender, poolId, stakeId);
}

function compound(uint256 poolId, uint256[] calldata stakeIds) external
isUser(msg.sender) isPool(poolId) nonReEntrant {
    uint256 cnt = stakeIds.length;

    for (uint256 i; i < cnt; i++) { _compound(msg.sender, poolId, stakeIds[i]); }
}

function _compound(address user, uint256 poolId, uint256 stakeId) private {
    Pool storage pool = _pool[poolId];
    Stake storage stake = _user[user].pool[poolId][stakeId];

    require(stake.amount > 0, "Withdrawn");

    unchecked {
      require(stake.lockedUntil > _timestamp(), "Locked");

      uint256 rewards = calculateRewards(user, poolId, stakeId);

      if (pool.taxCompound > 0) {
        uint256 tax = _percentage(rewards, uint256(pool.taxCompound));

        rewards -= tax;
        pool.collectedTaxes += tax;

        if (_treasury != address(0)) {
          bool txFee = IERC20(pool.rewardToken).transfer(_treasury, tax);
          require(txFee, "Transfer error");
        }
      }
    ......................
      stake.amount += rewards;

      emit Compound(user, poolId, stakeId, rewards);
    }
}
```

- Recommendation:
  - Ensure that the corresponding taxes will never exceed 100% to prevent overflow.

FOUND THREATS

⚠ High Risk

If penalty tax is higher than the amount and/or claim tax is higher than rewards and has enough balances to transfer tax amount to the treasury wallet, overflow will occur in 'amount' and/or 'rewards' variables, causing user's stake and/or rewards amounts to be artificially inflated.

```
function _withdraw(address user, uint256 poolId, uint256 stakeId) private {
    Pool storage pool = _pool[poolId];
    Stake storage stake = _user[user].pool[poolId][stakeId];

    require(stake.amount > 0, "Withdrawn");

    uint256 amount = stake.amount;

    unchecked {
      if (stake.lockedUntil > _timestamp()) {
        if (!pool.allowEarlyUnstake) { revert("Locked"); }

        if (pool.earlyUnstakePenalty > 0) {
          uint256 penalty = _percentage(amount, uint256(pool.earlyUnstakePenalty));

          amount -= penalty;
          pool.collectedPenalties += penalty;

          if (_treasury != address(0)) {
            bool penaltyFee = IERC20(pool.stakedToken).transfer(_treasury, penalty);
            require(penaltyFee, "Transfer error");
          }
        }
      }

      uint256 rewards = calculateRewards(user, poolId, stakeId);

      if (pool.taxWithdraw > 0) {
        uint256 tax = _percentage(rewards, uint256(pool.taxClaim));

        rewards -= tax;
        pool.collectedTaxes += tax;

        if (_treasury != address(0)) {
          bool txFee = IERC20(pool.rewardToken).transfer(_treasury, tax);
          require(txFee, "Transfer error");
        }
      }

      .....................

      IERC20(pool.stakedToken).transfer(user, amount);
      IERC20(pool.rewardToken).transfer(user, rewards);

      emit Withdraw(user, poolId, stakeId, amount);
    }
}
```

```
function withdraw(uint256 poolId, uint256 stakeId) external isUser(msg.sender)
isPool(poolId) isStake(poolId, stakeId) nonReEntrant {
    _withdraw(msg.sender, poolId, stakeId);
}

function withdraw(uint256 poolId, uint256[] calldata stakeIds) external
isUser(msg.sender) isPool(poolId) nonReEntrant {
    uint256 cnt = stakeIds.length;

    for (uint256 i; i < cnt; i++) { _withdraw(msg.sender, poolId, stakeIds[i]); }
}
```

- Recommendation:
  - Ensure that the corresponding taxes will never exceed 100% to prevent overflow.

Informational

Owner can open/close existing staking pools.

```
function setPoolStatus(uint256 poolId, bool open) external
onlyOwner isPool(poolId) {
    _pool[poolId].open = open;

    if (open) {
      emit PoolOpened(poolId);
    } else {
      emit PoolClosed(poolId);
    }
}
```

Owner can change existing staking pool's APY.

```
function setPoolAPY(uint256 poolId, uint24 apy) external onlyOwner isPool(poolId) {
    require(_pool[poolId].totalStaked == 0, "Pool active");
    require(apy > 0, "Invalid APY");

    _pool[poolId].apy = apy;

    emit APYChanged(poolId, apy);
}
```

Owner can change existing staking pool's rewards claim period.

```
function setPoolClaimCooldown(uint256 poolId, uint32 time) external onlyOwner isPool(poolId) {
    _pool[poolId].claimCooldown = time;

    emit ClaimCooldownChanged(poolId, time);
}
```

Informational

Owner can change existing pool's lockup status and lockup period.

```
function setPoolLockupPeriod(uint256 poolId, uint32 time) external onlyOwner isPool(poolId) {
    _pool[poolId].lockupPeriod = time;

    emit LockupPeriodChanged(poolId, time);
}

function setPoolLockupEnabled(uint256 poolId, bool enabled) external onlyOwner isPool(poolId) {
    _pool[poolId].lockupEnabled = enabled;

    emit LockupEnabledChanged(poolId, enabled);
}
```

Owner can change existing staking pool's reward period.

```
function setPoolRewardPeriod(uint256 poolId, uint32 time) external onlyOwner isPool(poolId) {
    _pool[poolId].rewardPeriod = time;

    emit RewardPeriodChanged(poolId, time);
}
```

Owner can change staking pool's early unstake status.
When early unstake is enabled users can unstake at any time.
Note: Early unstake taxes may apply.

```
function setPoolAllowEarlyUnstake(uint256 poolId, bool status) external onlyOwner isPool(poolId) {
    _pool[poolId].allowEarlyUnstake = status;

    emit AllowEarlyUnstakeChanged(poolId, status);
}
```

Informational

Owner can change staking pool's deposit, claim, compound, withdraw and early unstake taxes.

```
function setPoolTaxes(uint256 poolId, uint24 taxDeposit, uint24 taxClaim,
uint24 taxCompound, uint24 taxWithdraw) external onlyOwner isPool(poolId) {
    _pool[poolId].taxDeposit = taxDeposit;
    _pool[poolId].taxClaim = taxClaim;
    _pool[poolId].taxCompound = taxCompound;
    _pool[poolId].taxWithdraw = taxWithdraw;

    emit TaxesChanged(poolId, taxDeposit, taxClaim, taxCompound, taxWithdraw);
}

function setPoolEarlyUnstakePenalty(uint256 poolId, uint24 percent) external onlyOwner isPool(poolId) {
    _pool[poolId].earlyUnstakePenalty = percent ;

    emit EarlyUnstakePenaltyChanged(poolId, percent);
}
```

Owner can change staking pool's minimum and maximum stake amount per user.

```
function setPoolStakeAmount(uint256 poolId, uint256 minStake, uint256 maxStake) external onlyOwner isPool(poolId) {
    require(maxStake == 0 || maxStake > minStake);

    _pool[poolId].minStake = minStake;
    _pool[poolId].maxStake = maxStake;

    emit StakeAmountChanged(poolId, minStake, maxStake);
}
```

- Recommendation:
  - Ensure that taxes will never exceed 100% to prevent overflow in other functions.

Informational

Owner can withdraw any tokens from the contract.
When this function is present, in cases tokens sent into the contract by mistake or purposefully, contract's owner can retrieve them.

```
function withdrawNative(address payable to, uint256 amount) external payable onlyOwner {
    require(amount > 0);
    require(address(this).balance >= amount, "Insufficient balance");

    (bool success, ) = to.call{ value: amount }("");
    require(success);

    emit WithdrawnNative(to, amount, msg.sender);
}

function withdrawERC20(address token, address to, uint256 amount) external onlyOwner {
    require(amount > 0);
    require(IERC20(token).balanceOf(address(this)) >= amount, "Insufficient balance");

    bool success = IERC20(token).transfer(to, amount);
    require(success);

    emit WithdrawnERC20(token, to, amount, msg.sender);
}
```

SPYWOLF

CRYPTO SECURITY

ABOUT US

We are a growing crypto security agency offering audits, KYCs and consulting services for some of the top names in the crypto industry.

- **OVER 700 SUCCESSFUL CLIENTS**
- **MORE THAN 1000 SCAMS EXPOSED**
- **MILLIONS SAVED IN POTENTIAL FRAUD**
- **PARTNERSHIPS WITH TOP LAUNCHPADS, INFLUENCERS AND CRYPTO PROJECTS**
- **CONSTANTLY BUILDING TOOLS TO HELP INVESTORS DO BETTER RESEARCH**

Audits | KYCs | dApps
Contract Development

To hire us, reach out to
contact@spywolf.co or
t.me/joe_SpyWolf

FIND US ONLINE

SPYWOLF.CO

@SPYWOLFNETWORK

@SPYWOLFNETWORK

Disclaimer

This report shows findings based on our limited project analysis, following good industry practice from the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, overall social media and website presence and team transparency details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report.
While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER:
By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice.
No one shall have any right to rely on the report or its contents, and SpyWolf and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (SpyWolf) owe no duty of care towards you or any other person, nor does SpyWolf make any warranty or representation to any person on the accuracy or completeness of the report.
The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and SpyWolf hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, SpyWolf hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against SpyWolf, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts, website, social media and team.
No applications were reviewed for security. No product code has been reviewed.